ALCF Datascience frameworks: Tensorflow, PyTorch, Keras, and Horovod

Huihuo Zheng
Data science group
huihuo.zheng@anl.gov

Outline

- Datascience modules on Theta
  - How we built Tensorflow, Pytorch, Keras, and Horovod
  - Best practices for using the modules / installing other python packages
  - OMP threading and environmental variables setup (Tensorflow)
- Data distribution parallelization with Horovod and Cray ML plugin
- Visualization through Tensorboard
- Profiling using timeline trace and Vtune

“datascience” modules on Theta

```
[user@thetalogin4 ~]$ module avail datascience

-------------- /soft/environment/modules/modulefiles ---------------
datascience/horovod-0.13.11      datascience/tensorflow-1.4
datascience/keras-2.2.2          datascience/tensorflow-1.6
datascience/pytorch-0.5.0-mkldnn datascience/tensorflow-1.10
datascience/tensorboard
```

- Specifically optimized for KNL(*CPU*), with AVX512
- Using intel python 3.5 (based on intelpython35 module)
- GCC/7.3.0
- With –g, could be used for profiling
- Linked to MKL and MKL-DNN (home build)
- Dynamically linked to external libraries (be careful of your LD_LIBRARY_PATH, PYTHONPATH)
- With MPI through Horovod

Contact me huihuo.zheng@anl.gov if you find any issues.

How to use the "datascience" modules

- The packages are compiled with AVX512 vectorization, <u>it does NOT run directly on login nodes or mom nodes.</u>

```
>>> import tensorflow as tf
2018-09-23 20:08:54.289480: F tensorflow/core/platform/cpu_feature_guard.cc:37]
The TensorFlow library was compiled to use AVX512F instructions, but these aren'
t available on your machine.
Aborted (core dumped)          Or Illegal instruction (core dumped)
```

- Run with qsub script.sh, or on mom node interactively through aprun.

```
#!/bin/bash
#COBALT -A SDL_Workshop
#COBALT -n 128
#COBALT -q default --attrs mcdram=cache:numa=quad

module load datascience/tensorflow-1.10 datascience/horovod-0.13.11 datascience/keras-2.2
aprun -n nproc -N nproc_per_node -cc depth -j 2 python script
```

How to use the “datascience” modules

- We suggest you DO NOT use virtual environment. If your applications need other custom python packages, **pip install the package** to a separate directory and add the path to PYTHONPATH:

```
> module load intelpython35 gcc/7.3.0
> pip install package_name --target=/path_to_install
> export PYTHONPATH=$PYTHONPATH:/path_to_install/
```

- Or you could try to **build your own package as follows:**

```
> module load intelpython35 gcc/7.3.0 datascience/tensorflow-1.10
> python setup.py build
> export PYTHONPATH= $PYTHONPATH:/path_to_install/lib/python3.5/site-packages
> python setup.py install --prefix=/path_to_install/
```

In some cases, you might need to run “*aprun -n 1 -cc none python setup.py build*”

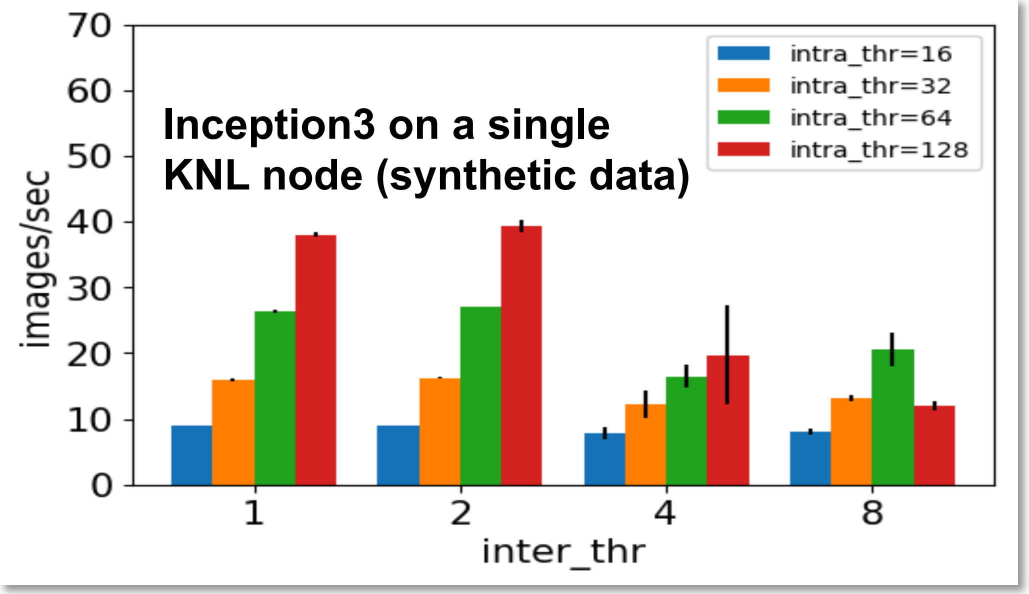
- Other suggestions:
  - unset PYTHONPATH and LD_LIBRARY_PATH, and then load “datascience” modules.
  - Always check currently loaded modules: “module list”
  - Always load datascience modules after you have loaded other modules.
  - Do not install packages to .local/lib/python3.5/site-packages (~~*pip install XXX --user*~~)

Tensorflow threading and OMP environmental variables

- **Thread placement in Tensorflow**
  - **inter_op_parallelism_threads**: Number of thread teams for executing different operations concurrently.
  - **intra_op_parallelism_threads**: The total number threads in the threads pool. This value should equal to **OMP_NUM_THREADS**
- **Threading setup**

```
config = tf.ConfigProto()
config.intra_op_parallelism_threads = num_intra_threads
config.inter_op_parallelism_threads = num_inter_threads
tf.Session(config=config)
```

- Optimal setup on Theta based on benchmarks
  - inter_op_parallelism_threads = 1, 2
  - Intra_op_parallelism_threads = OMP_NUM_THREADS (64*j/ppn)
  - Use aprun -e OMP_NUM_THREADS=.. to setup threads
  - aprun -j 2 is slightly better than aprun –j 1

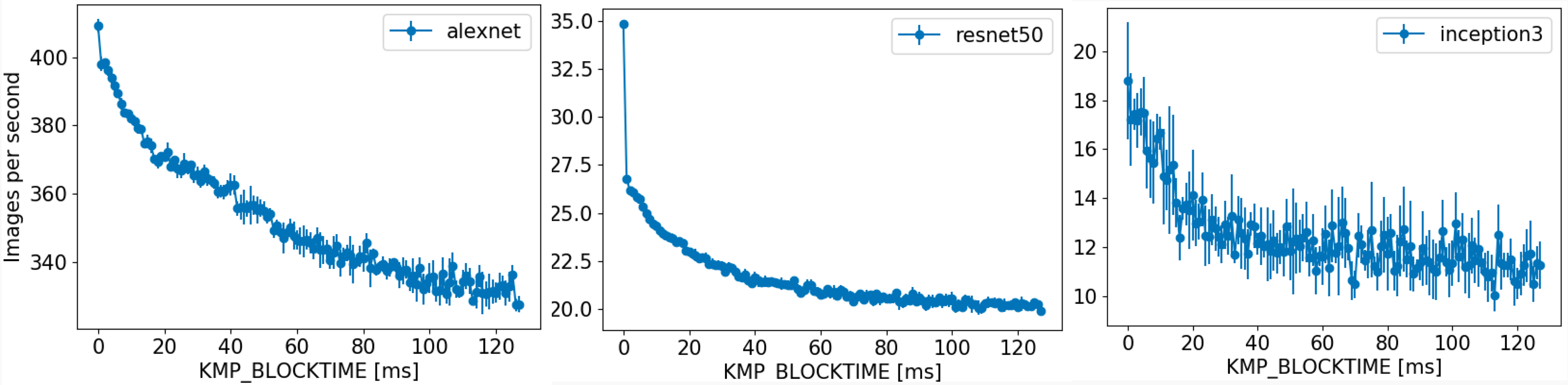


Tensorflow thread performance benchmarks

https://github.com/tensorflow/benchmarks/blob/mkl_experiment/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py

Tensorflow threading and OMP environmental variables

`KMP_BLOCKTIME=0`

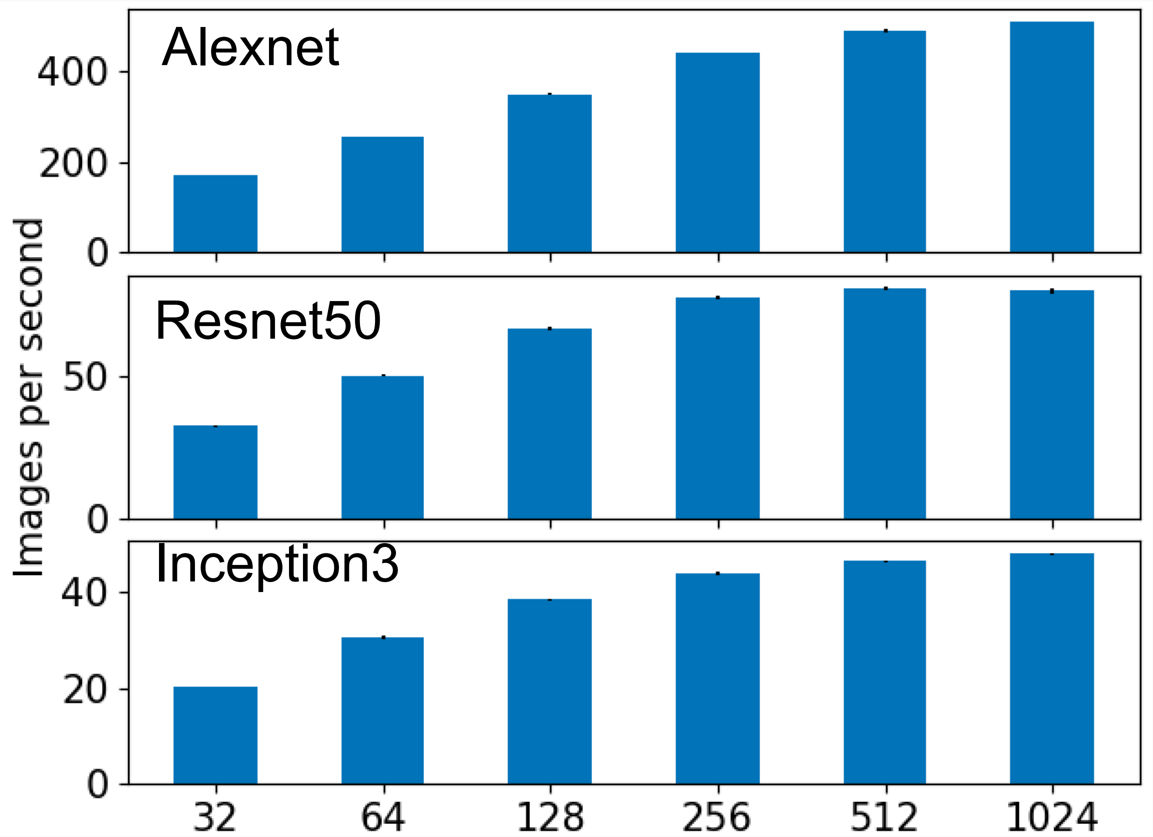


- The MKL default is 200ms, which was not optimal in our testing.
- You could set KMP_BLOCKTIME in two ways:
  1. Parse through aprun: aprun ... -e KMP_BLOCKTIME=0 …
  2. Set inside your python script: os.environ['KMP_BLOCKTIME']=0

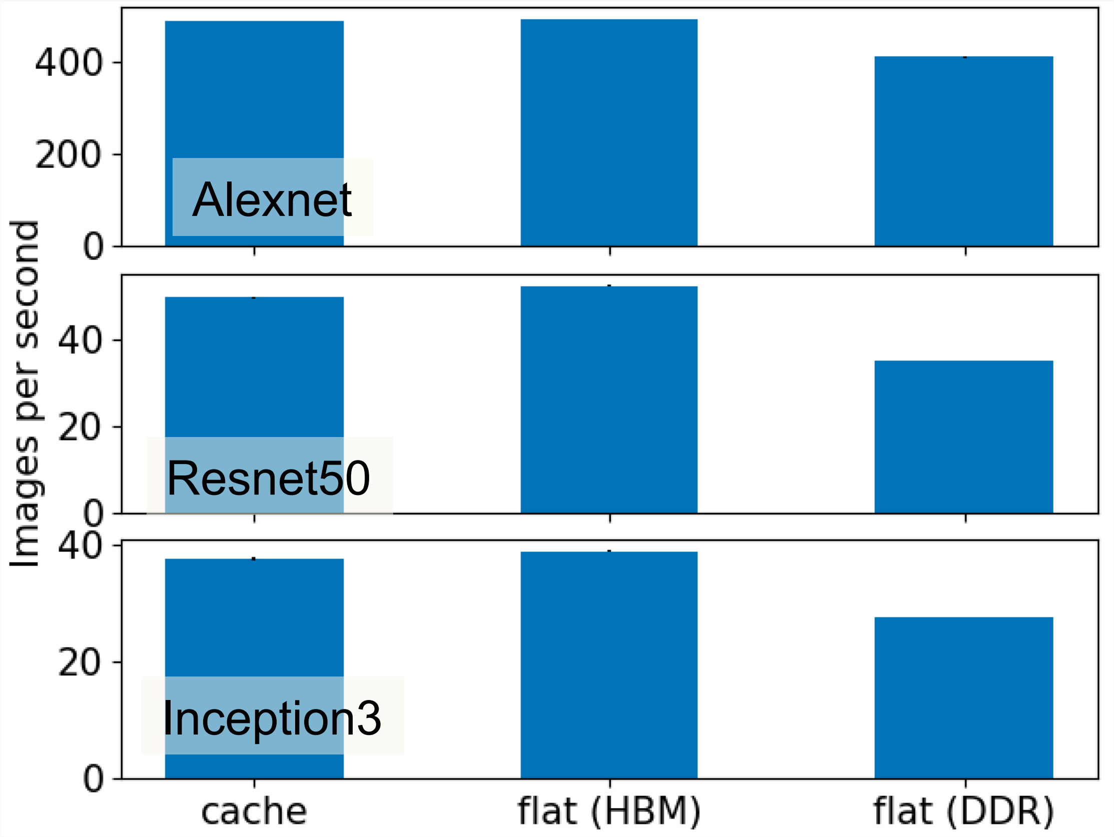
```
KMP_AFFINITY=granularity=fine,verbose,compact,1,0
```

If you set "aprun … -cc depth …", it automatically sets KMP_AFFINITY.

Batch size and memory mode (Tensorflow)

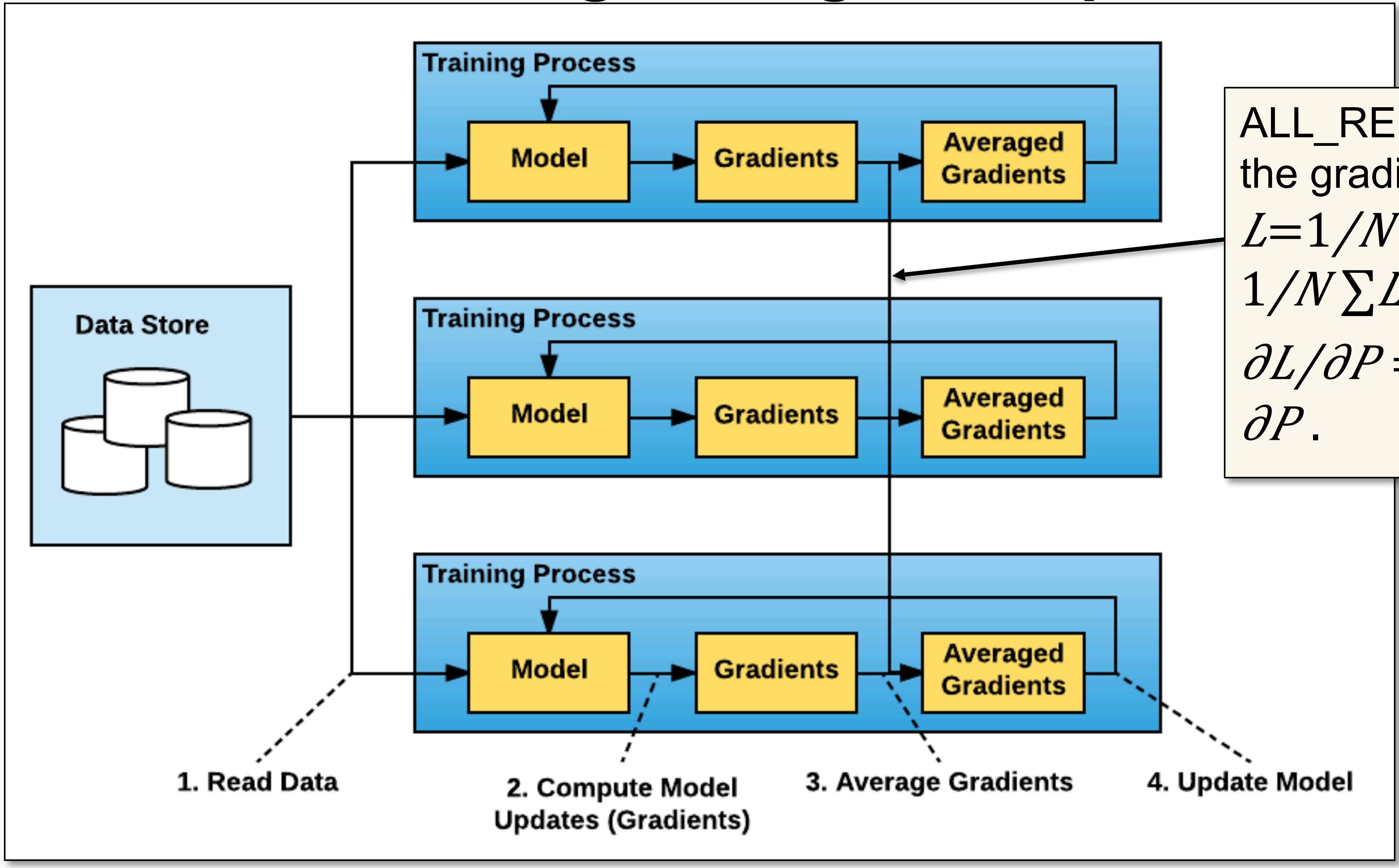


Batch size dependence



Different memory modes

Distributed learning through data parallelization



ALL_REDUCE of loss and the gradient

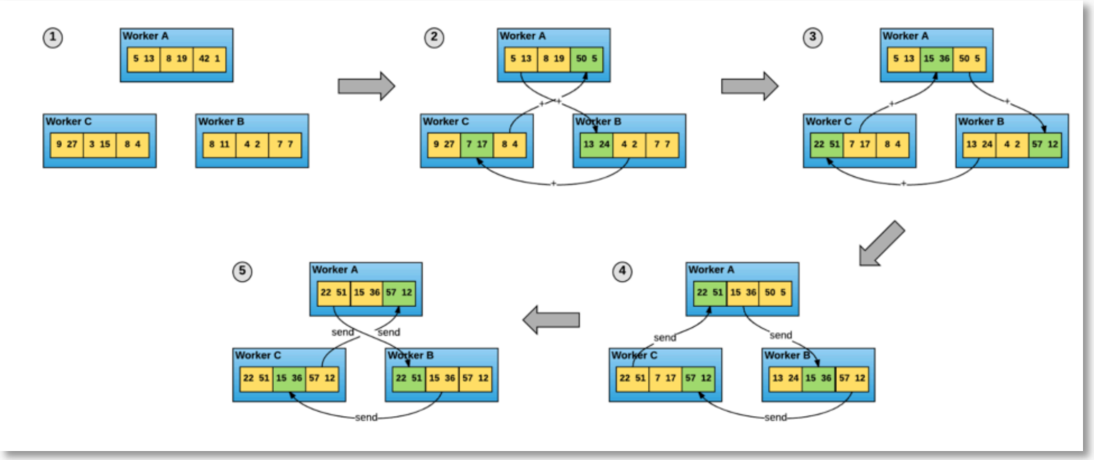
$$L = \frac{1}{N} \sum_i F(P, X_i) = \frac{1}{N} \sum L_i .$$

$$\frac{\partial L}{\partial P} = \frac{1}{N} \sum_i \frac{\partial L_i}{\partial P} .$$

https://eng.uber.com/horovod/

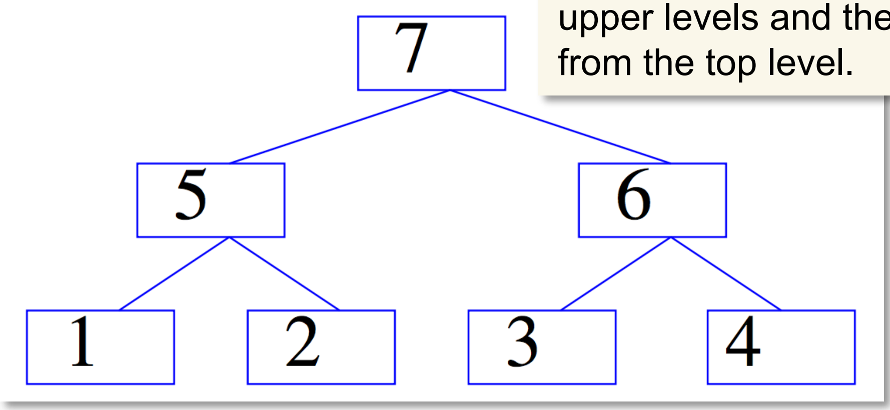
All Reduce in HOROVOD

- **Ring all reduce**



- Simultaneously utilize all the network connection.
- The message communicated each time is M/ nproc. (potentially becomes latency bound → fusion)

- **Tree all reduce**



Data flow from the lower levels to the upper levels and then broadcasted from the top level.

- Message size is larger. However, not all the network connections are simultaneously utilized

Distributed learning with HOROVOD

- Load the module in your environment

```
> module load datascience/horovod-0.11.13
> module load datascience/keras-2.2 datascience/tensorflow-1.10
```

- Change your python script
  - Initialize horovod (similar to MPI_init)

```
import horovod.keras as hvd
hvd.init() # hvd.rank() – rank id; hvd.size() – total number of process
```

  - Wrap the optimizer with Distributed optimizer (adjust learning rate)

```
#Adjust the learning rate proportionally
opt = keras.optimizers.Adadelta(1.0 * hvd.size())
opt = hvd.DistributedOptimizer(opt)
```

  - Broadcast the model from rank 0, so that all the ranks have the same beginning

```
callbacks=[hvd.callbacks.BroadcastGlobalVariablesCallback(0)]
```

Tensorflow with HOROVOD

```
import tensorflow as tf
import horovod.tensorflow as hvd
layers = tf.contrib.layers
learn = tf.contrib.learn
def main():
   # Horovod: initialize Horovod.
   hvd.init()
   # Download and load MNIST dataset.
   mnist = learn.datasets.mnist.read_data_sets('MNIST-data-%d' % hvd.rank())
   # Horovod: adjust learning rate based on number of GPUs.
   opt = tf.train.RMSPropOptimizer(0.001 * hvd.size())
   # Horovod: add Horovod Distributed Optimizer
   opt = hvd.DistributedOptimizer(opt)
   hooks = [
        hvd.BroadcastGlobalVariablesHook(0),
        tf.train.StopAtStepHook(last_step=20000 // hvd.size()),
        tf.train.LoggingTensorHook(tensors={'step': global_step, 'loss': loss},
                                   every_n_iter=10),
    ]
   checkpoint_dir = './checkpoints' if hvd.rank() == 0 else None
   with tf.train.MonitoredTrainingSession(checkpoint_dir=checkpoint_dir,
                                          hooks=hooks,
                                          config=config) as mon_sess
```

https://github.com/uber/horovod/blob/master/examples/tensorflow_mnist.py

PyTorch with HOROVOD

```
#…
import torch.nn as nn
import horovod.torch as hvd
hvd.init()
train_dataset = datasets.MNIST('data-%d' % hvd.rank(), train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,))
                   ]))
train_sampler = torch.utils.data.distributed.DistributedSampler(
    train_dataset, num_replicas=hvd.size(), rank=hvd.rank())
train_loader = torch.utils.data.DataLoader(
    train_dataset, batch_size=args.batch_size, sampler=train_sampler, **kwargs)
# Horovod: broadcast parameters.
hvd.broadcast_parameters(model.state_dict(), root_rank=0)
# Horovod: scale learning rate by the number of GPUs.
optimizer = optim.SGD(model.parameters(), lr=args.lr * hvd.size(),
                      momentum=args.momentum)
# Horovod: wrap optimizer with DistributedOptimizer.
optimizer = hvd.DistributedOptimizer(
    optimizer, named_parameters=model.named_parameters())
```

https://github.com/uber/horovod/blob/master/examples/pytorch_mnist.py

Keras with HOROVOD

```
import keras
import tensorflow as tf
import horovod.keras as hvd
# Horovod: initialize Horovod.
hvd.init()
# Horovod: adjust learning rate based on number of GPUs.
opt = keras.optimizers.Adadelta(1.0 * hvd.size())
# Horovod: add Horovod Distributed Optimizer.
opt = hvd.DistributedOptimizer(opt)
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=opt,
              metrics=['accuracy'])
callbacks = [
    # Horovod: broadcast initial variable states from rank 0 to all other processes.
    hvd.callbacks.BroadcastGlobalVariablesCallback(0),
]
# Horovod: save checkpoints only on worker 0 to prevent other workers from corrupting them.
if hvd.rank() == 0:
    callbacks.append(keras.callbacks.ModelCheckpoint('./checkpoint-{epoch}.h5'))
model.fit(x_train, y_train, batch_size=batch_size,
          callbacks=callbacks,
          epochs=epochs,
          verbose=1, validation_data=(x_test, y_test))
```

https://github.com/uber/horovod/blob/master/examples/keras_mnist.py

Cray ML Plugin

Check - Mike Ringenburg's talk: Scaling Deep Learning Frameworks (Cray)

- **Module setup**

```
module load cray-python/3.6.1.1
module load /lus/theta-fs0/projects/SDL_Workshop/mendygra/tmp_inst/modulefiles/craype-ml-plugin-py3/1.1.0
export PYTHONUSERBASE=/lus/theta-fs0/projects/SDL_Workshop/mendygra/pylibs
```

```
Example script: $CRAYPE_ML_PLUGIN_BASEDIR/examples/tf_mnist/mnist.py
Look for "CRAY ADDED" region
```

- **Initialization**

```
# initialize the Cray PE ML Plugin (assume 20M variables max)
mc.init(1, 1, 20*1024*1024, "tensorflow")
# config the thread team (correcting the number of epochs for the effective batch size))
FLAGS.train_epochs = int(FLAGS.train_epochs / mc.get_nranks())
max_steps = int(math.ceil(FLAGS.train_epochs * (_NUM_IMAGES['train'] +
_NUM_IMAGES['validation']) / FLAGS.batch_size))
mc.config_team(0, 0, 100, max_steps, 2, 200) # give each rank its own directory to save in
FLAGS.model_dir = FLAGS.model_dir + '/rank' + str(mc.get_rank())
```

- **Finalization**

```
mc.finalize()
```


Peter Mendygral's Slides: https://www.alcf.anl.gov/files/mendygral_ALCF_ScalingDL.pdf

Cray ML Plugin

- **Update optimizer to synchronize and apply**

```
if FLAGS.enable_ml_comm:
# we need to split out the minimize call below so we can
modify gradients
    grads_and_vars = optimizer.compute_gradients(loss)
    grads = mc.gradients([gv[0] for gv in grads_and_vars],
    0)
    gs_and_vs = [(g,v) for (_,v), g in zip(grads_and_vars,
    grads)]
    train_op = optimizer.apply_gradients(gs_and_vs,
    # END CRAY ADDED
    global_step=tf.train.get_or_create_global_step())
```

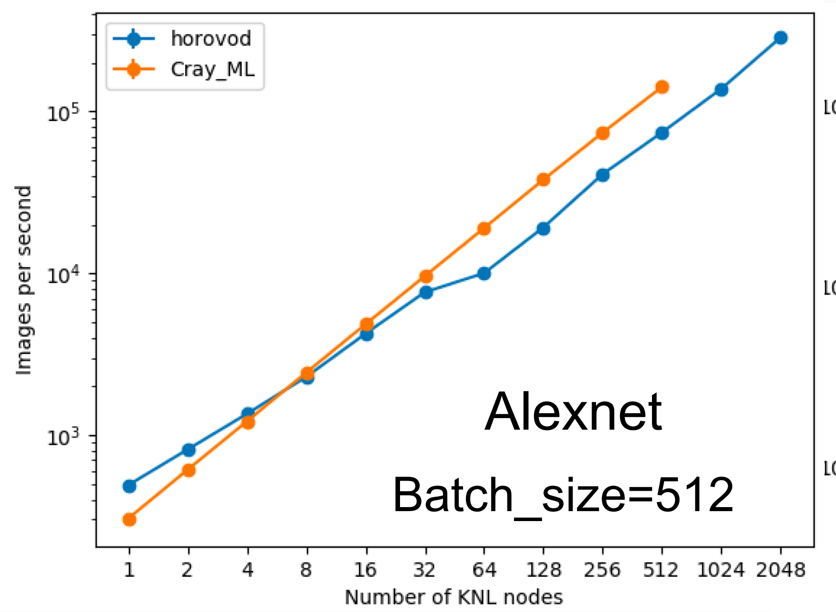
Cray ML Plugin

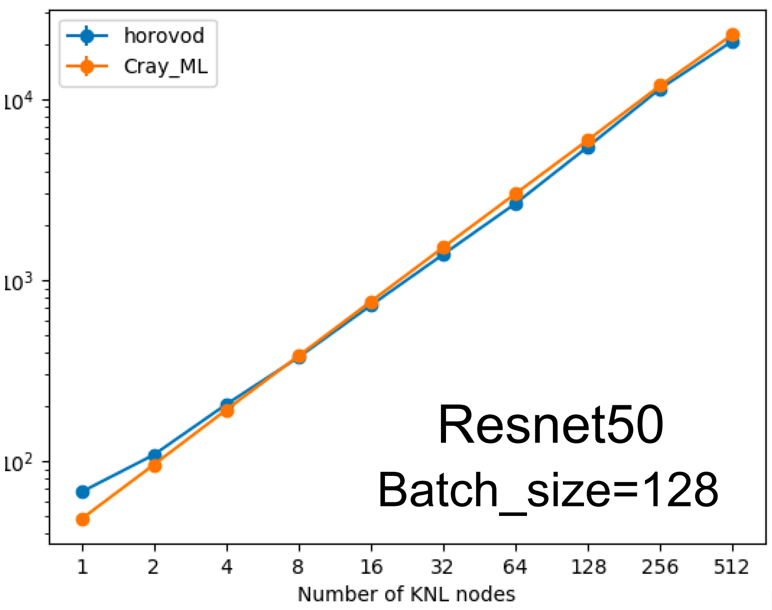
- **Create a hook to initialize variables**

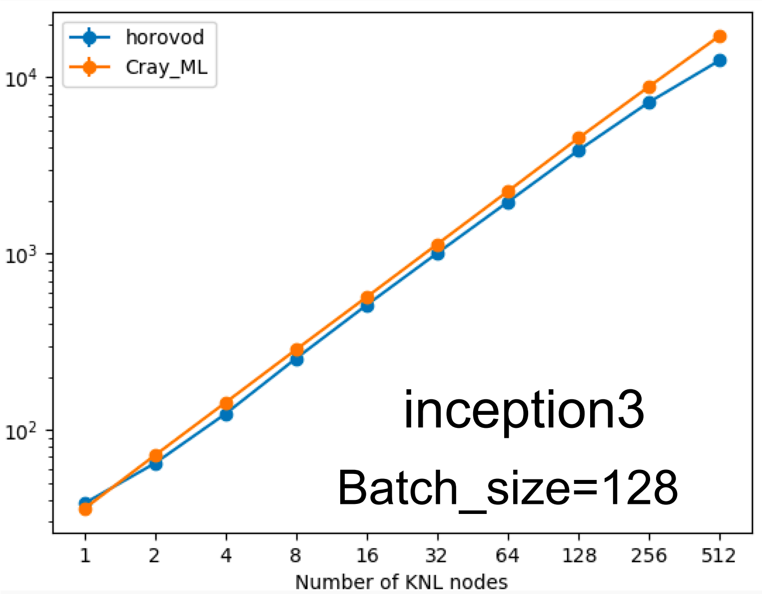
```
class BcastTensors(tf.train.SessionRunHook):
    def __init__(self): self.bcast = None
    def begin(self):
    if not self.bcast:
        new_vars = mc.broadcast(tf.trainable_variables(),0)
        self.bcast = tf.group(*[tf.assign(v,new_vars[k]) for k,v in
        enumerate(tf.trainable_variables())])
    def after_create_session(self, session, coord):
        session.run(self.bcast)
        if FLAGS.ml_comm_validate_init:
           py_all_vars = [session.run(v) for v
        tf.trainable_variables()]
        if (mc.check_buffers_match(py_all_vars,
          print("ERROR: not all processes have
        model!")
        else:
          print("Initial model is consistent on
```

```
sess_hooks = []
if FLAGS.enable_ml_comm:
    sess_hooks = [BcastTensors()] # END CRAY ADDED
# ...
tf.estimator.EstimatorSpec(
    mode=mode,
    predictions=predictions,
    loss=loss, train_op=train_op,
    training_hooks=sess_hooks,
    eval_metric_ops=metrics)
```

Scaling Tensorflow: HOROVOD / Cray ML Plugin (Synthetic data)



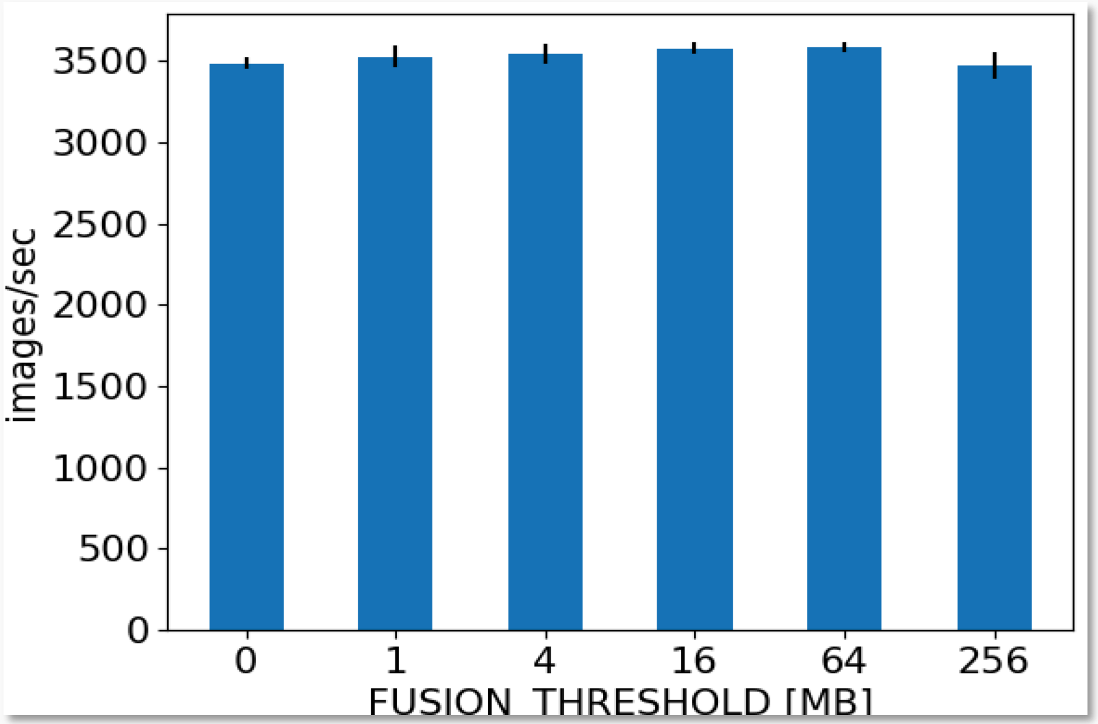




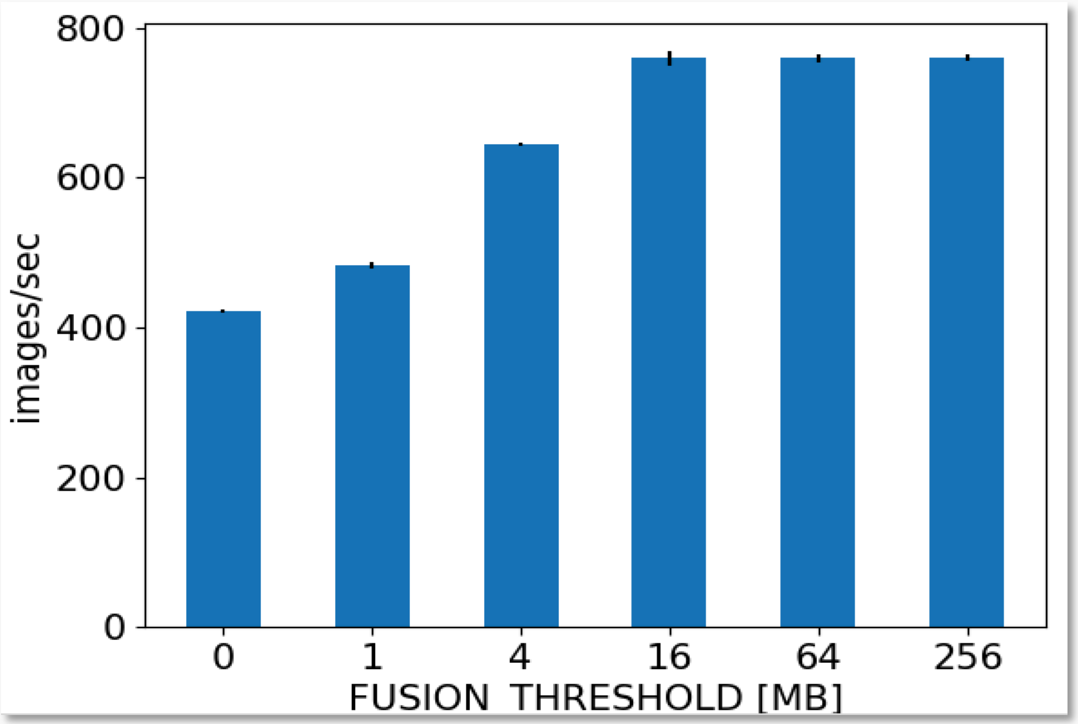
Cray ML plugins has better scaling efficiency than horovod. *[The fact that Cray ML plugin in1KNL case is slower than horovod is probably due to different tensorflow builds (1.10 intel vs 1.5 cray)]*

HOROVOD environmental variables: FUSION_THRESHOLD (default: 64MB)

Alexnet (16 KNL)



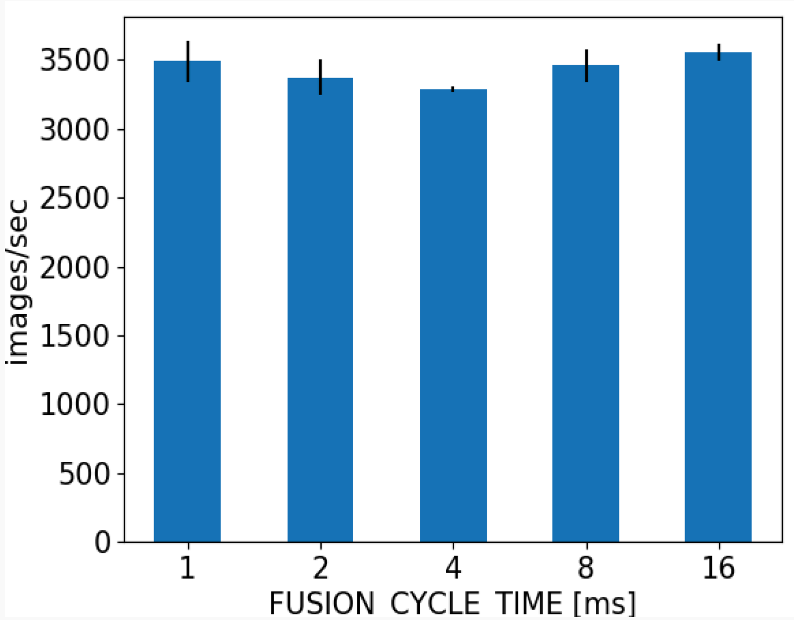
Inception3 (16 KNL)



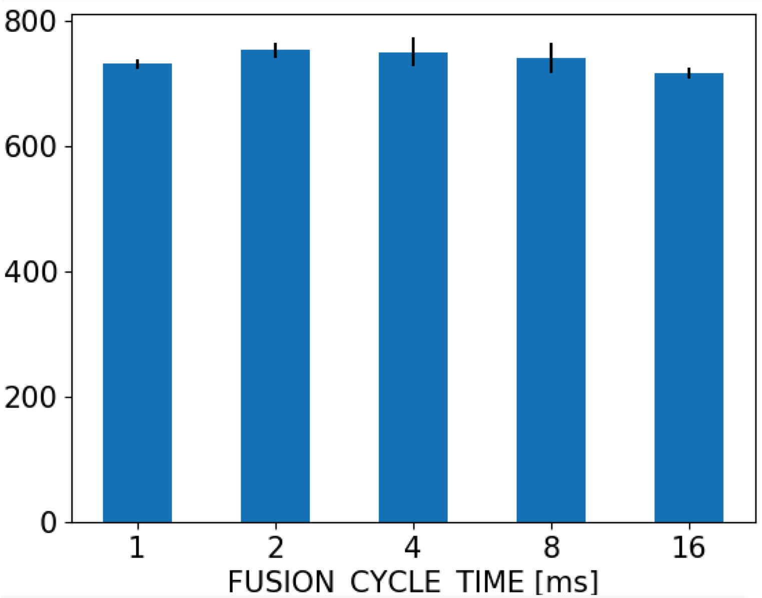
FUSION_THRESHOLD = 64 MB already gets optimal performance.

HOROVOD environmental variables: FUSION_CYCLE_TIME (default: 3.5ms)

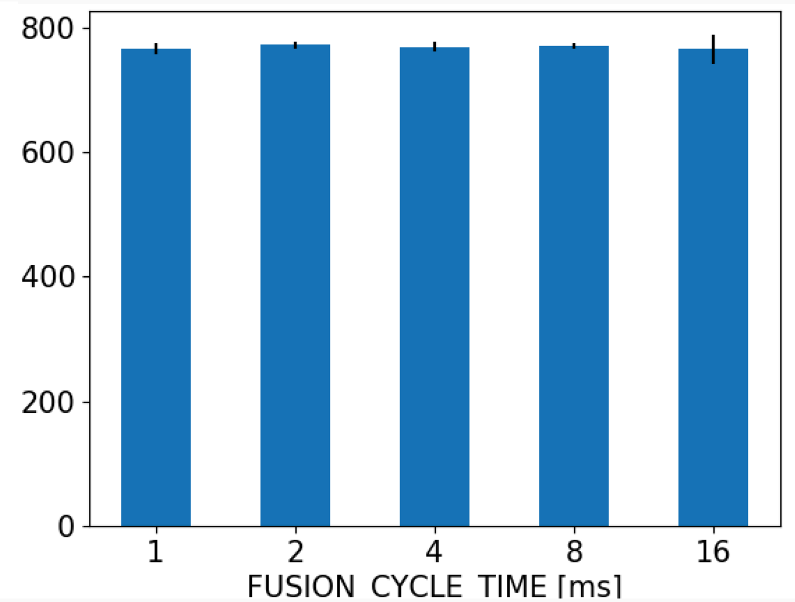
AlexNet (16 KNL)



ResNet50 (16 KNL)



Inception3 (16 KNL)



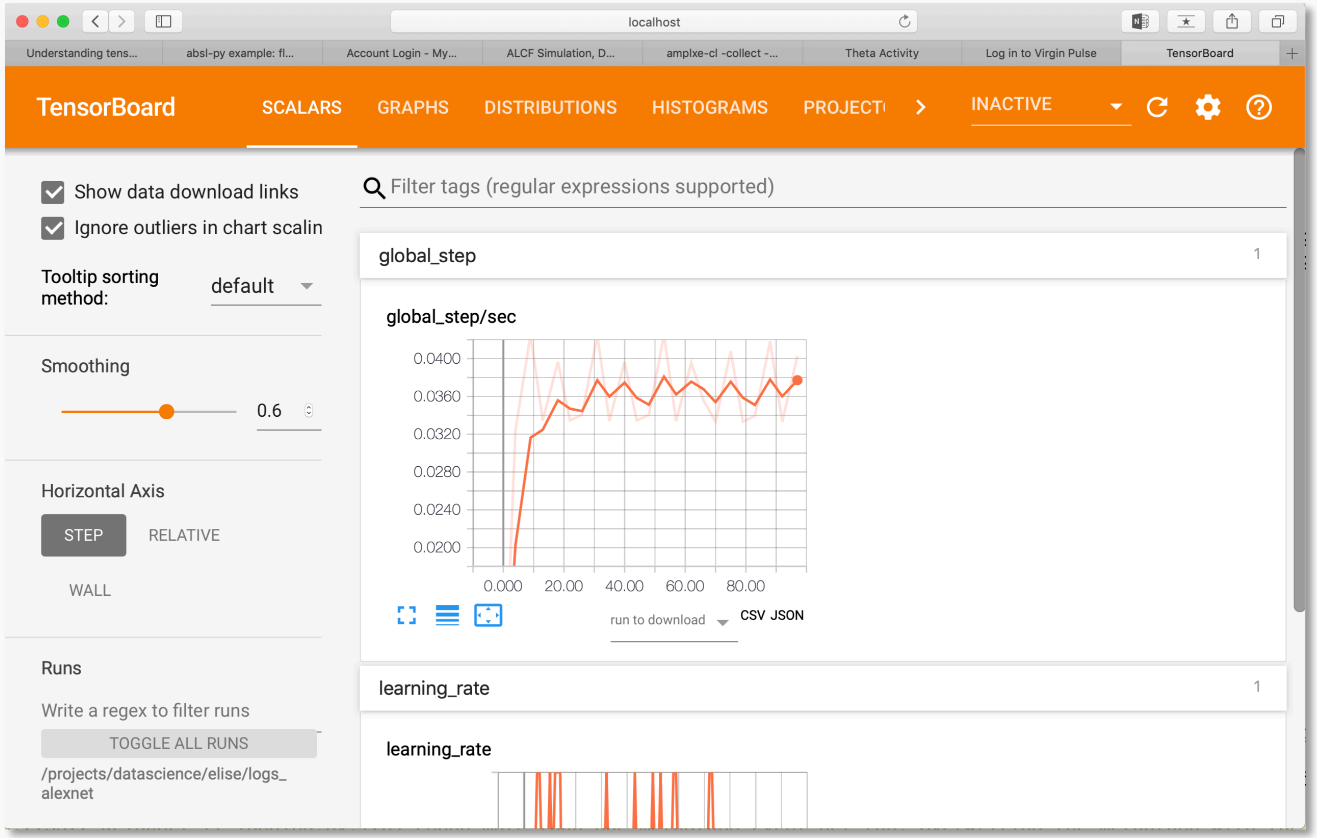
The runtime is not sensitive with respect to the changing of FUSION_CYCLE_TIME

Comments about distributed deep learning

Increasing workers increases the global batch size and the learning rate:

- This reduces the number of updates to the model (iterations) per epoch
- Might require more iterations to converge to same validation accuracy;
- Might have different convergence;
- Might need warm up steps with smaller learning rate.

Visualization with Tensorboard



Read log files through ssh tunneling

(1) SSH tunnel to Theta

ssh -XL 16006:127.0.0.1:6006 user@theta.alcf.anl.gov

(2) Run tensorboard on Theta

> module load tensorboard

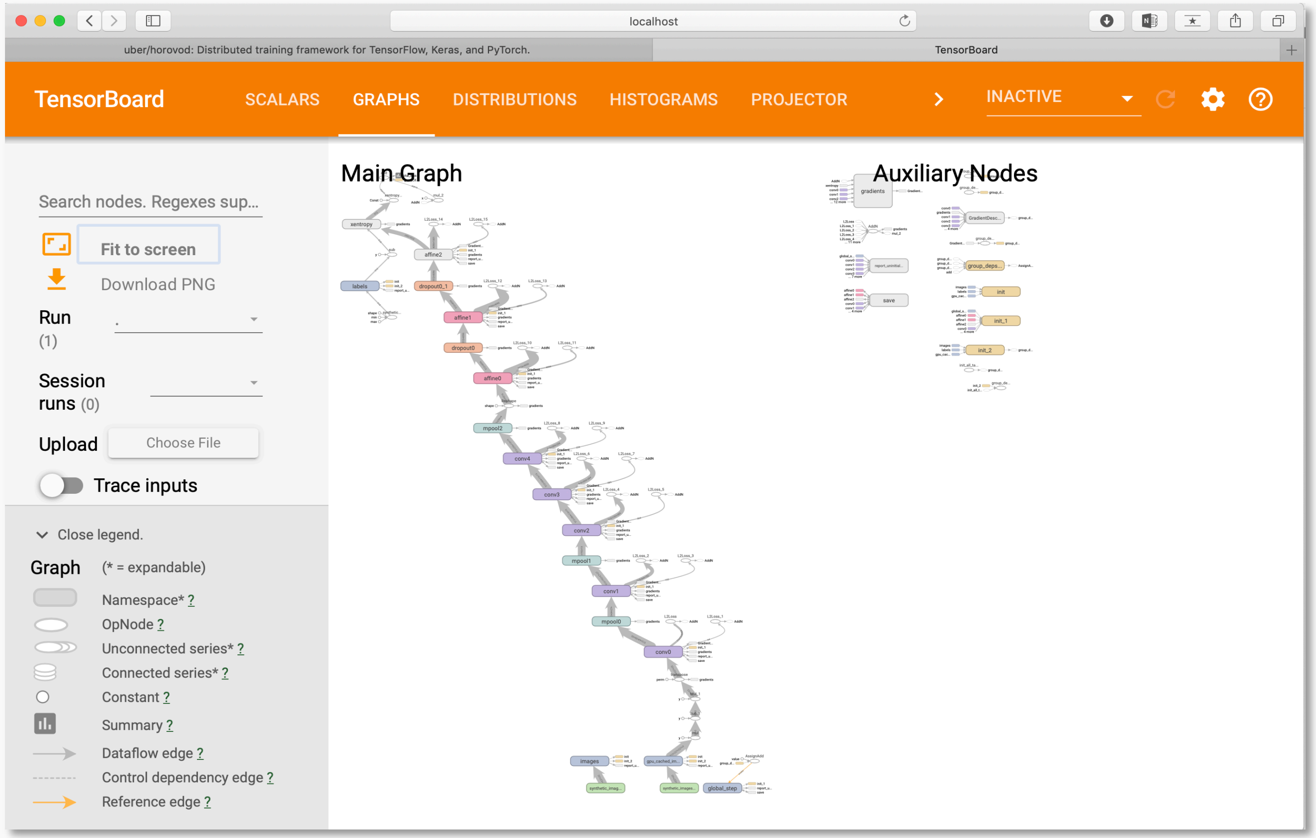
> tensorboard --logdir DIR

(3) Open browser from local machine: https://localhost:16006

Interactive job controlling through Tensorboard is not supported on Theta yet.

https://www.datacamp.com/community/tutorials/tensorboard-tutorial

Visualization with Tensorboard



Read log files through ssh tunneling

(1) SSH tunnel to Theta

ssh -XL 16006:127.0.0.1:6006 user@theta.alcf.anl.gov

(2) Run tensorboard on Theta

> module load tensorboard

> tensorboard --logdir DIR

(3) Open browser from local machine: https://localhost:16006

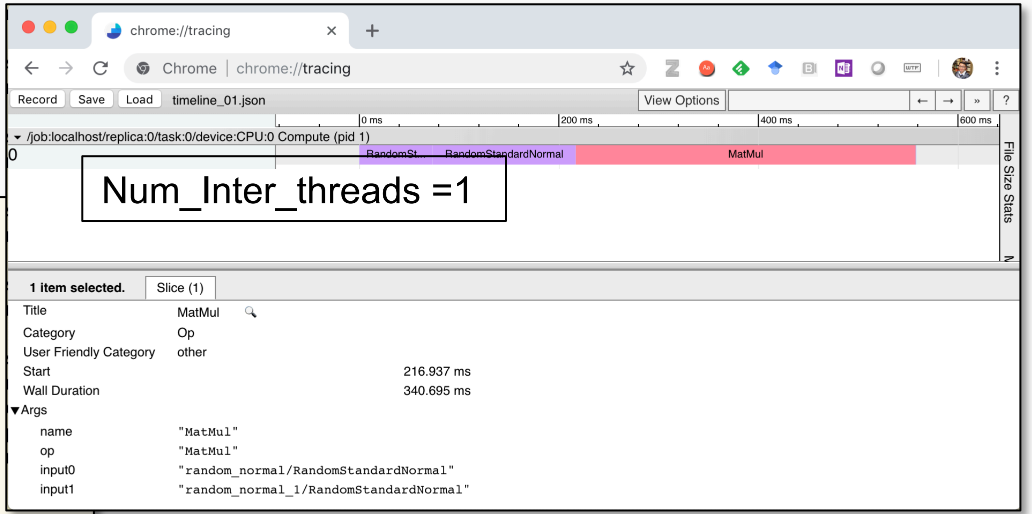
Interactive job controlling through Tensorboard is not supported on Theta yet.

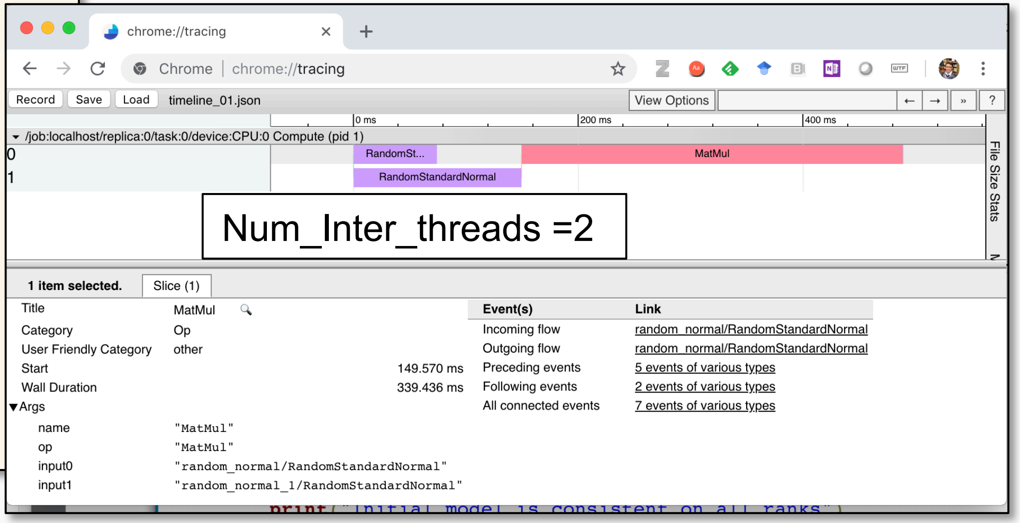
https://www.datacamp.com/community/tutorials/tensorboard-tutorial

Argonne NATIONAL LABORATORY

Tracing profile

```
import tensorflow as tf
from tensorflow.python.client import timeline
import sys
a = tf.random_normal([2000, 5000])
b = tf.random_normal([5000, 1000])
res = tf.matmul(a, b)
sess = tf.Session(config=tf.ConfigProto(\
        inter_op_parallelism_threads=1,\
            intra_op_parallelism_threads=1 ))
# add additional options to trace the session execution
options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRACE)
run_metadata = tf.RunMetadata()
sess.run(res, options=options, run_metadata=run_metadata)
# Create the Timeline object, and write it to a json file
fetched_timeline = timeline.Timeline(run_metadata.step_stats)
chrome_trace = fetched_timeline.generate_chrome_trace_format()
f=open('timeline_01.json', ’w’); f.write(chrome_trace);f.close()
```

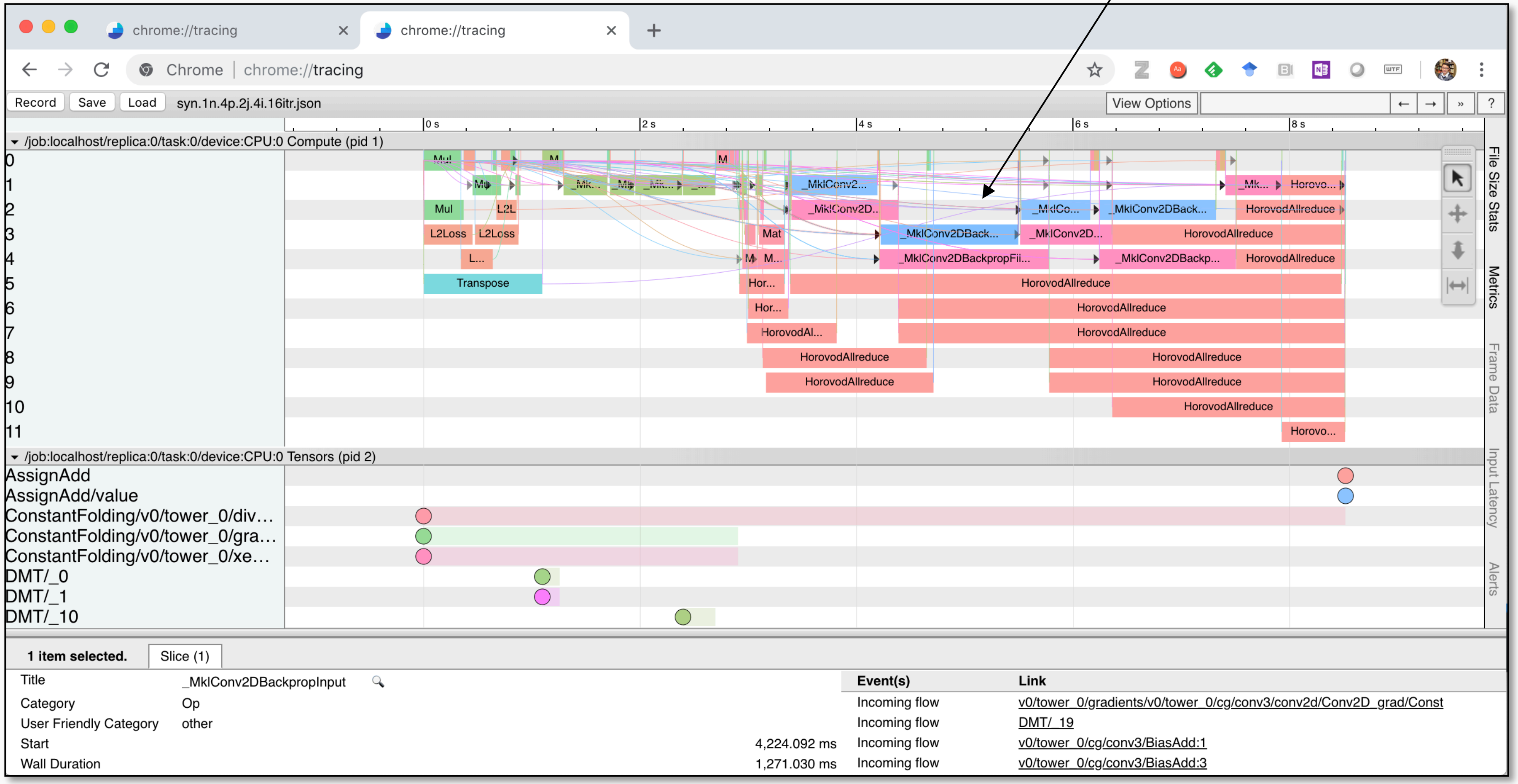




Open timeline_01.json using Chrome.
Go to the page chrome://tracing. ”Load” the JSON file.

Tracing profile (Alexnet)

Dataflow



Time spent on different kernels (Alexnet)

294 items selected. Slices (294)

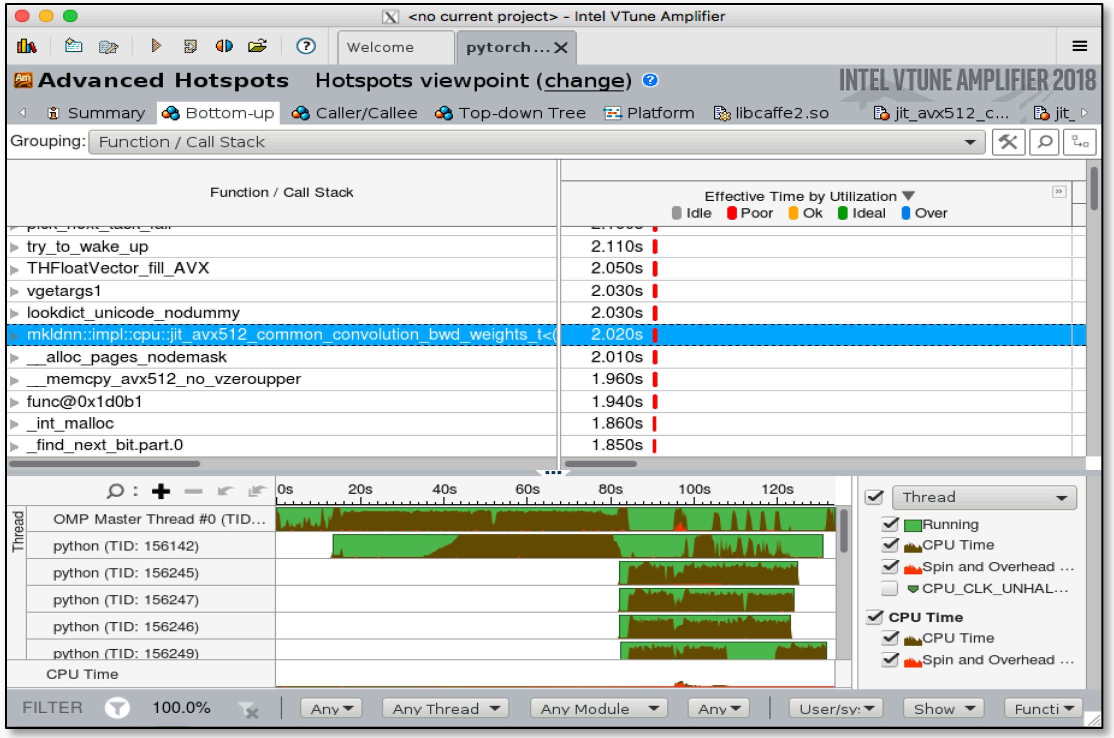
Name	Wall Duration	Self time	Average Wall Duration	Occurrences
_MklConv2DBackpropFilterWithBias	1,680.151 ms	1,680.151 ms	336.030 ms	5
_MklConv2DBackpropInput	884.677 ms	884.677 ms	221.169 ms	4
_MklConv2DWithBias	791.824 ms	791.824 ms	158.365 ms	5
Transpose	555.096 ms	555.096 ms	555.096 ms	1
MatMul	308.802 ms	308.802 ms	34.311 ms	9
_MklAddN	45.058 ms	45.058 ms	2.816 ms	16
_MklReluGrad	43.725 ms	43.725 ms	6.246 ms	7
_MklRelu	34.106 ms	34.106 ms	4.872 ms	7
BiasAddGrad	29.678 ms	29.678 ms	9.893 ms	3
_MklMaxPoolGrad	28.922 ms	28.922 ms	9.641 ms	3
Mul	21.469 ms	21.469 ms	0.613 ms	35
_MklMaxPool	16.323 ms	16.323 ms	5.441 ms	3
ApplyGradientDescent	10.671 ms	10.671 ms	0.667 ms	16
L2Loss	8.933 ms	8.933 ms	0.558 ms	16
SparseSoftmaxCrossEntropyWithLogits	7.694 ms	7.694 ms	7.694 ms	1
BiasAdd	2.790 ms	2.790 ms	0.930 ms	3
Const	2.061 ms	2.061 ms	0.027 ms	75
_MklReshape	1.587 ms	1.587 ms	0.794 ms	2
_MklToTf	0.864 ms	0.864 ms	0.041 ms	21
VariableV2	0.609 ms	0.609 ms	0.034 ms	18
Identity	0.370 ms	0.370 ms	0.019 ms	20
_MklIdentity	0.353 ms	0.353 ms	0.035 ms	10
NoOp	0.125 ms	0.125 ms	0.031 ms	4
RandomUniformInt	0.124 ms	0.124 ms	0.124 ms	1

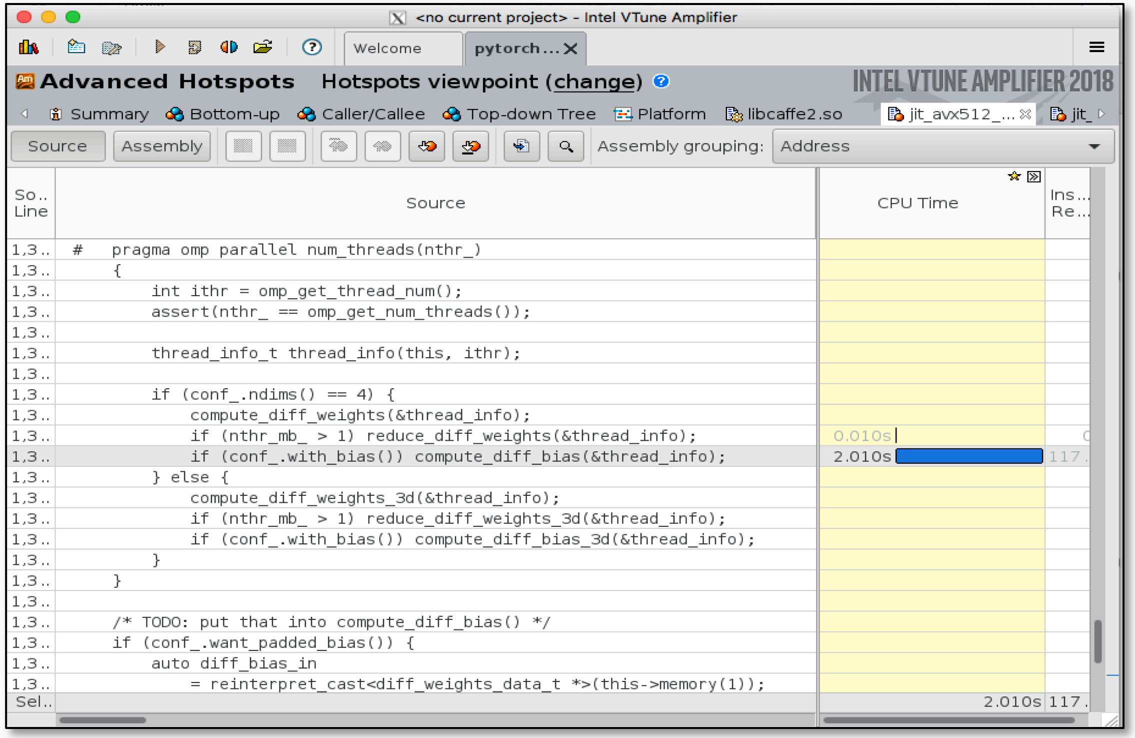
VTune profiling

More details: Profiling Your Application with Intel VTune and Advisor - Carlos Rosales-Fernandez and Paulius Velesko, Intel

Remember to set LD_LIBRARY_PATH,
Put vtune library at the end!! Otherwise, it might complaint about the GLIBCXX version.

```
source /opt/intel/vtune_amplifier/amplxe-vars.sh
aprun -n … -e OMP_NUM_THREADS=128 \
        -e LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/intel/vtune_amplifier/lib64 \
        ampxle-cl -collect advance-hotspots -r output_dir python script.py
```





The python modules are compiled using -g flag. Therefore, the user could trace the source file in Vtune.

Argonne NATIONAL LABORATORY

Thank you!

huihuo.zheng@anl.gov